International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V71I8P221

Volume 71 Issue 8, 243-252, August 2023

Original Article

Honeypot-Based Thread Detection using Machine Learning Techniques

Diandra Amiruddin Firmansyah[1], Amalia Zahra[2]

[1,2]*Department of Computer Science, BINUS Graduate Program, Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia.*

[1]*Corresponding Author : diandra.firmansyah@binus.ac.id*

Received: 23 May 2023 Revised: 28 June 2023 Accepted: 22 July 2023 Published: 15 August 2023

Abstract *- This paper explores the application of machine learning techniques to honeypot-based thread detection in cybersecurity. Honeypot is a decoy system designed to lure attackers and gather information about their methods and objectives. Honeypot-based thread detection is a proactive approach to cybersecurity that can identify and prevent attacks before they cause damage. However, the sheer volume of data generated by honeypots can be overwhelming for human analysts. In this context, machine learning techniques can help automate the analysis of honeypot data and improve threat detection accuracy. The performance is evaluated using real-world honeypot data. Based on the experiment results, the Random Forest algorithm demonstrated superior performance compared to other algorithms, with an accuracy rate of 99.20% for detecting malware. The results show that machine learning can significantly enhance the effectiveness of honeypot-based thread detection, enabling cybersecurity analysts to identify and respond to threats more quickly and efficiently.*

Keywords *- Malware, Machine learning, Honeypot.*

1. Introduction

Malicious software is now a significant risk to computer systems. They have brought about casualties and monetary losses. With the use of various methods, such as obfuscation, malicious individuals enhance the amount of malware[1].

Based on information published by Gdata, cyber attackers unleashed nearly 135,000 fresh versions of their harmful software, equivalent to more than 93 assault efforts per minute. The attackers are relying on the incapacity of antivirus programs to match the rapid pace and their inability to recognize the newly created malware variants, granting them unrestricted access to the network. Security experts detected over 49 million distinct malware applications in 2022, representing a surge of 107 percent, more than double the number recorded in 2021[2].

Table 1. Latest malware trend

Malware	Impact	Increment
Mobile banking Trojan	Fake updates, email, smishing	60%
Cryptocurrency Mining	Power drain, Overheating, Excessive Data Usage, Slow	80%
Ransomware	Encrypted data or pay the ransom	60%

Cybersecurity threats are causing significant damage to worldwide organizations and have increased exponentially[3]. Cybercriminals use various tactics such as malware, social engineering, and phishing to gain unauthorized access to sensitive data shown in Table 1. To minimize damage, such as in the data, initial prevention is needed to anticipate damage, such as detect cyber-attacks.

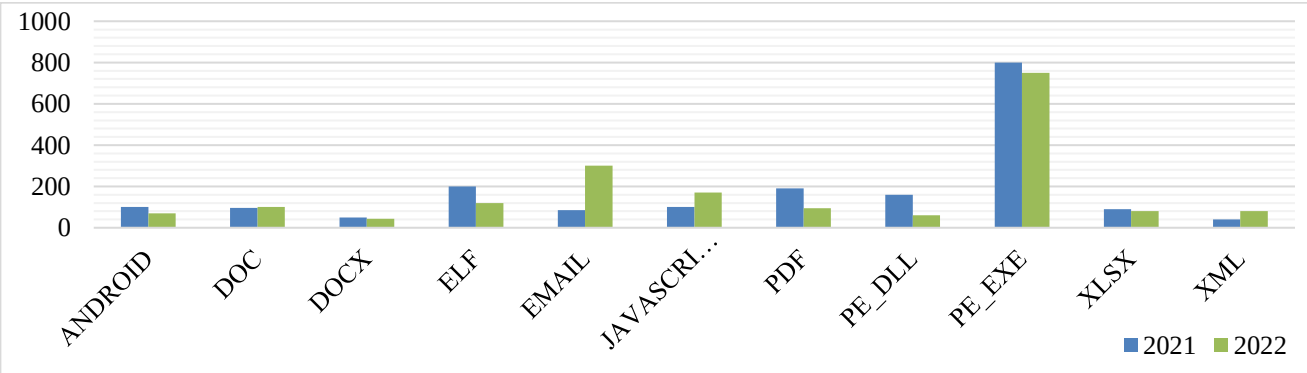


Fig. 1 Exploiting sample vulnerabilities by file type

According to statistics from VirusTotal, the PE file format is the most submitted[4]. Based on the data obtained from VirusTotal, the Portable Executable (PE) file format has highest number of submissions, as shown in Figure 1.

There are many ways to detect cyber-attacks, such as implementing security systems, log activity monitoring, firewall, intrusion detection system, penetration testing, honeypot, and machine learning. Overall, each technique of detecting cyber-attacks has its own advantages and disadvantages. Honeypots can be useful in detecting attacks before they happen, while other techniques such as firewalls, IDS, and machine learning can help detect attacks that have already occurred or are in progress. The appropriate detection technique depends on the system's specific needs and characteristics that need to be protected.

Honeypots serve as fake systems designed to attract threat actors, and they can be used to identify and evaluate attacks while also gathering data on the techniques and tools employed by the threat actor[5]. Machine learning is a subfield of artificial intelligence that enables computers to learn, adjust and adapt without explicit programming. Combining a honeypot with machine learning[6] can help identify and prevent cyber-attacks. Through the utilization of data collected from honeypots, machine learning algorithms can be trained to recognize patterns and abnormalities in network traffic that may signify the existence of an attack.

In recent times, there has been considerable interest in applying honeypots and machine learning for threat detection[7]. This technique has shown promise in detecting emerging threats that conventional signature-based detection systems may overlook.

Moreover, it can furnish immediate notification to security teams to react promptly and prevent the attacks from causing significant damage. A system for detecting malware using machine learning has been introduced. It utilizes the header of Portable Executable (PE) files to extract features and determine whether the file is clean or malicious. Additionally, it can detect zero-day malware in the file.

Previous studies have revealed that the average accuracy can be achieved up to 90%, with the evaluation results indicating the highest level of efficiency[8]. The task that will be conducted in this research is building a model for malware detection using machine learning.

Based on the state-of-the-art malware research, Random Forest, Decision Tree, and Support Vector Machine will be built to choose parameters from the dataset to improve better performance. Then the model will be constructed to classify the malware within the dataset. Finally, a machine learning model will be applied. This paper will utilize the strengths and advantages of the models mentioned above to enhance the performance of the malware detection model.

The proposed model employs a static analysis approach that involves four layers: data acquisition, preprocessing, performance evaluation, and prediction. This model can identify malware without the need to execute the executable file[23].

The approach is based on supervised machine learning techniques with binary classification to categorize the dataset into either malicious or benign classes. The performance of the proposed system is evaluated using various metrics, including accuracy, F1-score, precision, and recall. Based on the experimental results, the Random Forest algorithm demonstrated superior performance compared to other algorithms, achieving a 99.20% accuracy rate in detecting malware.

2. Related Works

Research related to malware classification has been developed using various combinations of algorithms with various datasets. The reason behind this is mainly because malicious code can often be designed in a manner that tricks detection tools. It is essential to employ effective protective measures to safeguard devices from the yearly surge of new malware. It is necessary to adopt more advanced security measures to protect data from potential harm caused by unfamiliar and dangerous ransomware threats[10].

A signature, like a fingerprint, is a distinctive characteristic or set of characteristics that uniquely distinguishes an executable. Typically, after installing an executable file with the belief that it is secure but contrary to expectations, it poses a significant threat. This is how malware infiltrates the system. Once on the screen, it spreads out and conceals itself within various files, making it challenging to detect. The aim is to obtain personal or valuable information; it can establish a direct connection to the operating system and initiate encryption[11].

Malware is one of the evolving dangers to information security, according to the study[12]. Malware detection is regarded as being beyond the capabilities of security solutions like antivirus, firewalls, and signature-based IDS. Recognizing new attack strategies, viruses, or worms outside of signature-based security systems is challenging.

Research conducted in [24] examined the exfiltration of private information by malicious software, with the goal of presenting a methodology for classifying and identifying malicious software to protect private information from malware threats. They employed data mining and machine learning classification methods by examining features based on signatures and anomalies. The research results show an accuracy of 98.6%.

In a different study, executable file types are used as the main input source for static malware detection[14]. This study focuses on malware detection and categorization. To detect behavior-based malware effectively, machine learning approaches need a dataset of malware with a variety of properties. The contribution of this study lies in the utilization of a minimal feature set for malware

identification. This approach involves using features determined based on the significance of rough sets, combined with Ant Colony Optimization (ACO) as a heuristic search technique. Malware can be discovered by analyzing portable executable (PE) files. A benchmark dataset for this work is a malware dataset called Classification of Malwares Portable Executable (ClaMP), containing integrated and raw features with an accuracy result of 95.02% for raw and 90.55% for integrated.

The study focused on static malware threats and used a behavioral-based detection approach[15]. This approach commonly uses frequency-based graph mining algorithms to extract unique patterns from a collection of malware graphs. It frequently uses graphs to describe system activities or behaviors. This paper proposes a new approach for malware detection, which involves utilizing compression-based mining on quantitative data flow graphs to generate a highly accurate detection model. This approach exceeds frequency-based detection models in terms of detection success according to a test that focused on a sizable and varied group of malwares. Using a machine-learning-enabled honeypot is one approach for detecting malware.

While machine learning can identify malware by classifying samples, honeypots can be employed to catch suspicious packets. The classification techniques employed include Decision Tree and Support Vector Machine (SVM).

3. Research Methodology

The topic of this research is to classify malicious software from the model that will be built with the dataset. This research proposed machine learning models to classify malware datasets. This research will be done in supervised learning as Random Forest, Decision Tree, and Support Vector Machine has already been labeled to train machine learning models.

This research begins by selecting the research topic and conducting a comprehensive review of relevant literature. Subsequently, the appropriate methodology and machine learning model are chosen. The dataset is then prepared, followed by data preprocessing and feature extraction. In the subsequent phase of the research, the model is fine-tuned and trained. Finally, after training all the models, they are evaluated using various metrics, including confusion matrix and F1-score. The experimental results are analyzed, and the conclusions are presented in the paper, as shown in Figure 2.

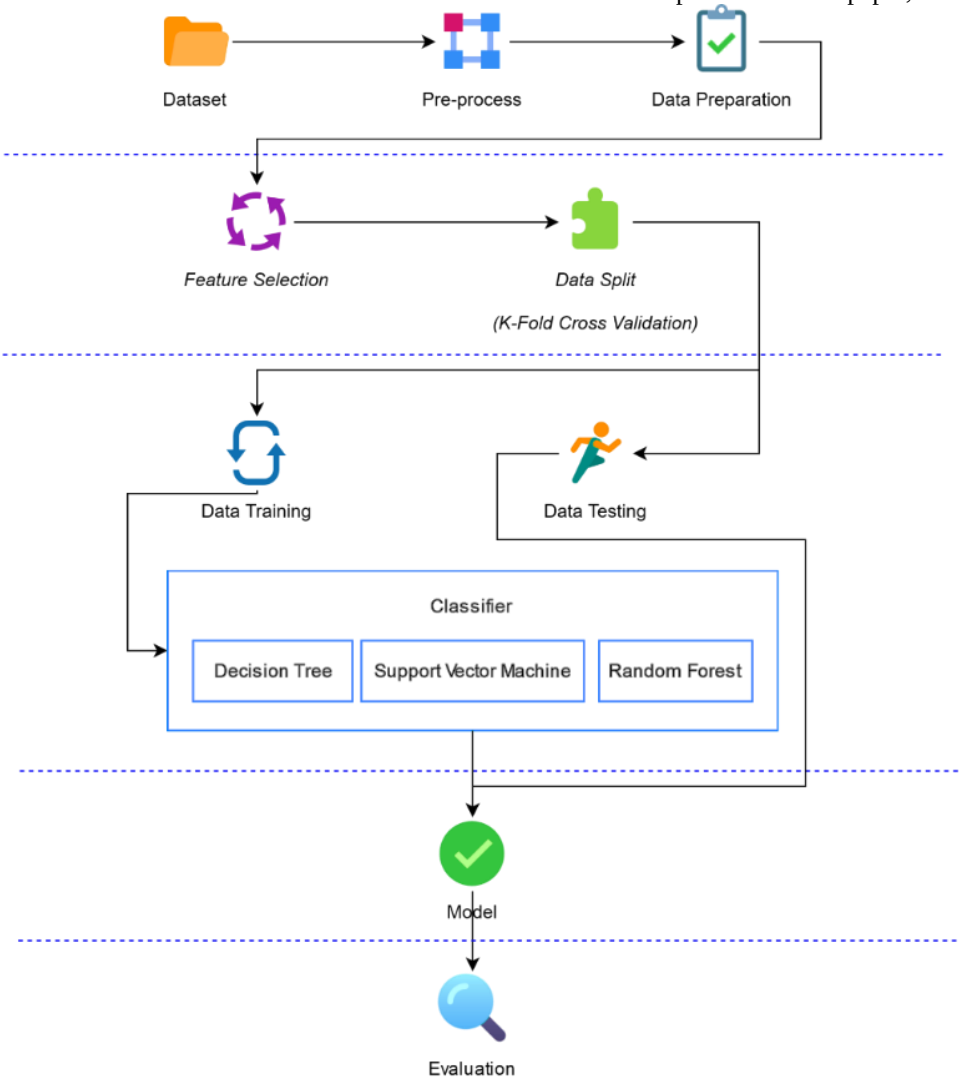


Fig. 2 Research process stages

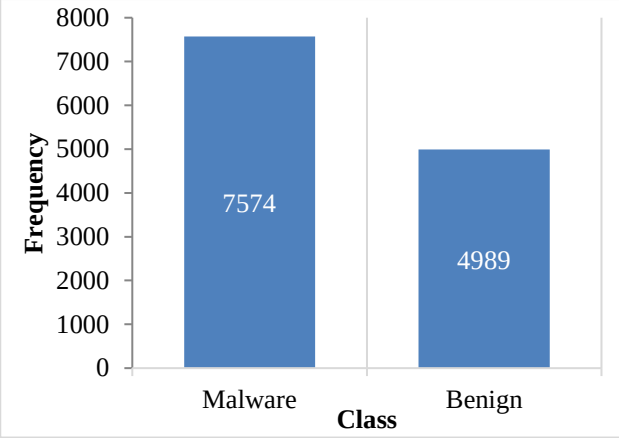


Fig. 3 Data distribution

3.1. Preparing Dataset

In this research, the Classification of Malwares Portable Executable (ClaMP) dataset will be utilized for training and building the model. ClaMP [16] contains 7574 *malware* and 4989 *benign*. The model's training involved using the Classification of Malware with PE headers (ClaMP) dataset, sourced from the Github repository.

The dataset was split into two groups—the first group comprised raw features [17], which consisted of 55 distinct characteristics. The second group was made up of integrated features, which included both derived and expanded features. However, in this research, the dataset used will be combined with malware data obtained from a Dionaea honeypot [18] that has been gathered from January until February 2023. The honeypot is installed in one of the government institutions in Indonesia.

The distribution of the malware in the dataset is shown the Figure 3. In this work, the data was obtained from a honeypot installed in one of the ministries in Indonesia and combined with an online data repository. The dataset contains header features of Portable Executable (PE) files and is categorized into benign and malicious classes. In the legitimate attribute of the dataset, the benign class is assigned a value of 0, while the malicious class is assigned a value of 1[25]. The header features used are 33 PE header features from a combination of data from an online repository and data recorded on the honeypot. Table 2 provides a description of the PE header, such as key features from PE header files and their corresponding descriptions.

Table 2. Portable executable header

Feature Name	Description
CheckSum	Only used for kernel driver/DLL mode and can be set to 0 for user executable/DLL mode.
Subsystem	The subsystem used by the file, with follower values specified in the file.
ImageBase	The first-bit address of the image file.
MajorimageVersion	The major/ primary version number of the image.
MinorImageVersion	Image minor version number.
SizeOfStackCommit	Amount of memory used on the stack.
AddressOfEntryPoint	RVA on *entry* point.
SectionAlignment	The sequence of loading the section into memory.
FileAlignment	The sequence of raw data sections in the image file.
SizeOfCode	Size of the code section.
SizeOfInitializedData	The size of the initialized data section.
NumberOfSections	Size of section table.
BaseOfData	Pointer when starting the data section.
BaseOfCode	Pointer when starting the code section.
SizeOfUninitializedData	Size of the uninitialized data section.
MajorOperatingSystemVersion	Referring to the major version number of the operating system.
MinorOperatingSystemVersion	Operating system minor version number.
SizeOfStackReserve	The value of the space allocated for the stack.
SizeOfImage	Image value.
MajorLinkerVersion	Number version Major Linker.
MinorLinkerVersion	Number version Minor Linker.
SizeOfHeapCommit	The number of bytes required for the input heap.
SizeOfHeaders	Header Size.
LoaderFlags	Reserved.
MajorSubsystemVersion	Major number version subsystem.
MinorSubsystemVersion	Minor number version subsystem.
SizeOfHeapReserve	Bytes received number by the heap.
e_cblp	Bytes on the last page of the file.
e_cp	Total number of pages that a file had.
e_cparhdr	Size of the header in paragraphs.
e_maxalloc	The maximum amount of additional allocation required.
e_sp	Value from sp register.
e_lfanew	Contain file address from exe header.

Preprocessing involves cleaning and filtering the data or files used as a dataset so that only the relevant data is processed further. The preprocessing is implemented by extracting signatures from each malware sample. Each malware sample will be identified by its MD5 hash [20] and recorded in a .csv file. The signatures are necessary for labeling in the next stage.

3.2. Feature Selection

The feature selection process involves choosing the variables, attributes, or subsets of variables pertinent to developing a model. To create sub-datasets, the method of data splitting is employed, where the data is divided into two or more parts. The dataset used for learning is known as data training. The modeling of the machine learning algorithm is done by applying supervised learning classification methods, as shown in Figure 4.

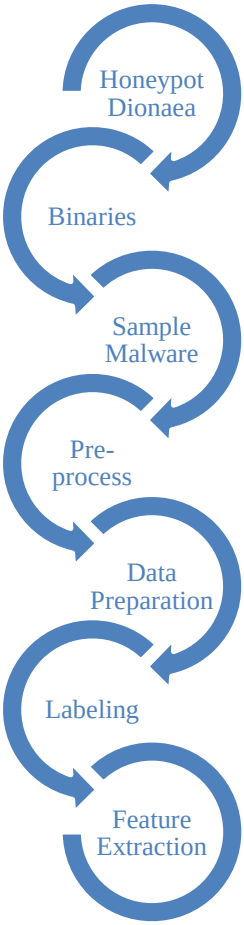


Fig. 4 Data gathering mechanism

3.3. Classifier

Data training assists machine learning modeling based on classification algorithms using separated data for training purposes. At the initial stage, the dataset is split into testing and training data, where X_train and y_train are variables used to implement the data training. X_train contains the Portable Executable (PE) features, while y_train is the class label that will be utilized for training the model. The result of the data training is saved in (.pkl) format and exported into a model format. The model file is then saved in the /model directory.

After completing the model training process using the training data, the separated testing data will be saved and used only for performance testing of the model. The testing data is part of the dataset that has been separated in the data-splitting phase and is used for evaluation after the model has been constructed. Implementation of the testing data is carried out by dividing the data into variables X_test and y_test. The variable X_test contains Portable Executable (PE) features, while y_test contains class labels.

3.4. Model

Only a portion of the data is used for training to build a model, and this data is also used to evaluate the model's performance. The dataset is split into two components: training data and testing data. The variation value of the data split is determined by the size of 90:10. Data splitting is implemented by separating the feature class into the variable y and other features into X. Then, each X and y is divided into 4 variables, namely X_train, X_test, y_train, and y_test.

3.5. Evaluation

The built model is tested to measure its performance. The evaluation was carried out on cross-validation, percentage split, and hyperparameters. In cross-validation, the data used is based on the training data, while in percentage split, the data used for evaluation is the testing data. Four classification metrics are used to evaluate accuracy, precision, recall, and f1-score[21]. In this study, the Primary dataset contains malware samples from the honeypot, and the Secondary data is the ClaMP dataset.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

$$\text{F1-Score} = 2 \text{ x } \frac{Precision \; x \; TPR}{Precision + TPR} \quad (4)$$

The accuracy is calculated by summing the number of true positives (TP) and true negatives (TN) and then dividing it by the total number. Equation 1 is utilized to compute the accuracy. Equation 2 is employed to compute precision. Precision represents the ratio of correctly predicted results to the total predicted results. It measures the relevance of the predicted results among all the predicted instances.

Recall, also known as sensitivity, represents the proportion of correctly predicted results to the actual results, indicating the accuracy of predicting the actual results. The formula in Equation 3 is utilized to calculate the recall. The F-measure, also referred to as the F1-score, is the harmonic means of precision and recall, as depicted in Equation 4. A high F-measure value close to 1 indicates an ML model with both high precision and recall, indicating its effectiveness and accuracy.

4. Result and Discussion

4.1. Preparing Dataset

The pefile library is used to read the header of the portable executable file so that feature extraction can be performed on malware samples[22]. The scikit-learn library splits the percentage of training and validation samples. The panda's library is used for data analysis and manipulation. From the malware obtained through the Dionaea Honeypot, there are at least 2169 file samples consisting of portable. The malware samples used in this study are portable executable types, including file extensions such as .exe, infected files in the .dll library extension, and .sys drivers.

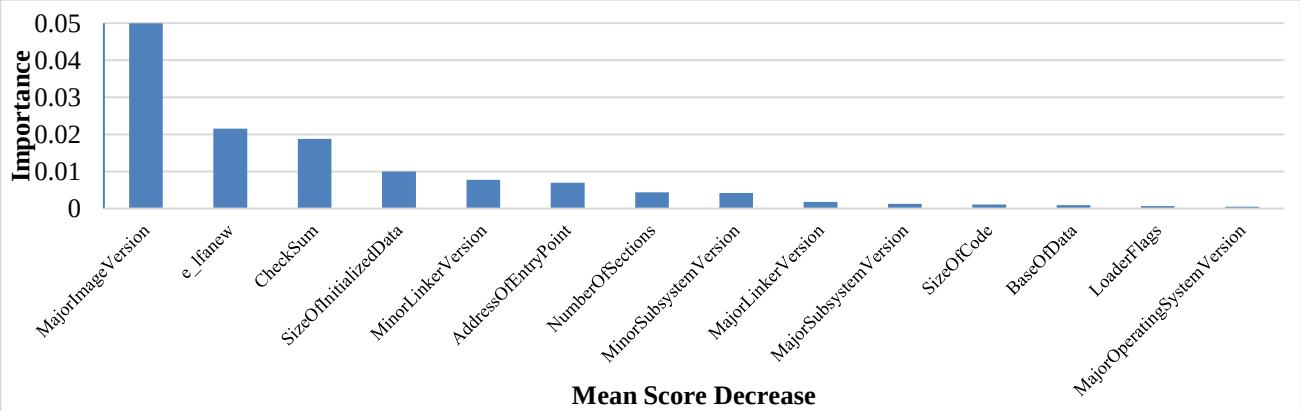
The dataset is examined for any unavailable values. Malware samples will be labeled by extracting signatures from each malware sample. MD5 hash identification is performed from the malware samples, as shown in Figure 5, and will be saved in .csv format. After obtaining the MD5 hash, the next process is to scan the hash using VirusTotal to ensure that the hash is malware or not.

4.2. Feature Selection

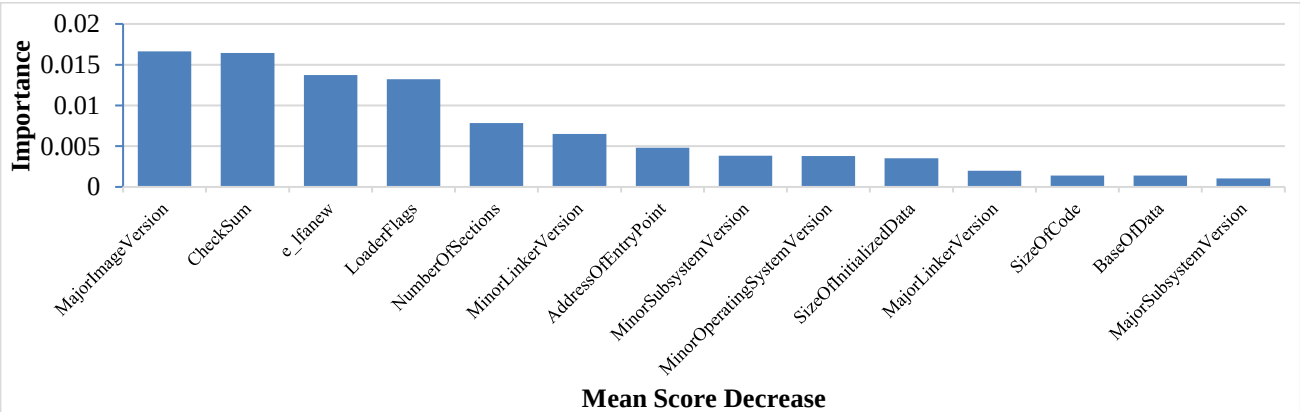
The feature selection phase involves the selection of relevant features for machine learning. In this stage, less relevant features are eliminated from the dataset to enhance the prediction performance and increase the efficiency of the proposed model. In this stage, the method used for feature selection is by using a tool called a wrapper. In each feature, an evaluation is performed using the Random Forest and Decision Tree algorithm to indicate which features impact accuracy most. The result from feature selection will apply to the model and improve the accuracy, as shown the Figure 6.

fileName	MD5hash
Trojan-Downloader.Win32.Small.bhp-9591dd13fb34e5b55aab16dafcc72ccb20d57dbe16f64581f38d08c9948ac037	a067199631132b7786d3d5340afd5509
Trojan-Downloader.Win32.Calac.ddw-0ad3d4fd88e6ebff5dff98d0eba46f1e8189b364e103fbf53f46b46b94479d6e	ff2d0dbb5935e0c0ce9fb81ce357e263
Trojan-Downloader.Win32.Delf.aznp-f61ebdb2aa82a9004ae2fc84d73e0d188253b3ad6f84608ddb71052a9f6ecd22	1dadb3157604f4f12e2451fca1b35875
Trojan-Downloader.Win32.CodecPack.ampn-009490744725fecf39986227f5742bc2eed500c674e39c7fa2b1f9b175882a15	2644aac529f8b901e5438c215a42635a
Trojan-Downloader.Win32.Delf.aznp-94563137393cbc39eb7dc51f7eabff825c467e4e1ba3303dfda8924fbd946ecd	17538f6adba9049f5983ae4efa852490
Trojan-Downloader.Win32.Adnur.wii-4f29836e87b058d9c2dda8d527affa2280400259915d5f465b3cb6d365290249	6f67f6e217da6fab829f4389fe40c035
Trojan-Downloader.Win32.Agent.gyha-2a89cfbedc934ba5bdabfca80e46ace310204389fb939f93f554b73ec96674e8	0851d8864f59902cb737a9193838e07e
Trojan-Downloader.Win32.Agent.xznf-1d00861ca70e65ff49213067979ad229f8fa086eae9abfccf53d40b1a33a7907	a5c4f128ea6a327aa0c48dde00352c24
Trojan-Downloader.Win32.Agent.aadcy-d9aeb50d89d618a2f0c2bbf372eea824d11e1ceab125ee03025636ad781da444	f402a837f00da3d39eb97d912897d63e
Trojan-Downloader.Win32.Genome.dryn-90c9a5495de923c700341be4becbfdcb01bfdab2e1100c70384b48fc6647dcac	695cc54df6bed2d258c401fee01ce8e1
Trojan-Downloader.Win32.Geral.anfr-5a3cda16058c909f5cc615aed2ff874cc0210b445b26347d51b8a3ba43eab668	4577f36271a4e6fc2aa49842f7a11bd7

Fig. 5. MD5 hash from portable executable file



(a) Measuring feature importance using random forest



(b) Measuring feature importance using decision tree

Fig. 6 Feature importance from the dataset

Table 3. Importance measure from each parameter

(Decision Tree)			(Random Forest)		
Feature	Importance	Stdev	Feature	Importance	Stdev)
CheckSum	0.2285	0.0035	MajorImageVersion	0.0499	0.0034
AddressOfEntryPoint	0.2080	0.0042	**e_lfanew**	0.0215	0.0009
SizeOfInitializedData	0.0993	0.0016	**CheckSum**	0.0187	0.0003
MinorLinkerVersion	0.0963	0.0013	**SizeOfInitializedData**	0.0100	0.0016
e_lfanew	0.0811	0.0021	**MinorLinkerVersion**	0.0077	0.0006
MajorLinkerVersion	0.0750	0.0022	**AddressOfEntryPoint**	0.0069	0.0012
SizeOfCode	0.0462	0.0014	NumberOfSections	0.0043	0.0002
MajorImageVersion	0.0399	0.0018	MinorSubsystemVersion	0.0042	0.0004
SizeOfStackReserve	0.0392	0.0008	MajorLinkerVersion	0.0017	0.0004
MajorSubsystemVersion	0.0368	0.0025	MajorSubsystemVersion	0.0012	0.0004
BaseOfData	0.0285	0.0012	SizeOfCode	0.0010	0.0002
NumberOfSections	0.0273	0.0027	BaseOfData	0.0009	0.0003
MinorSubsystemVersion	0.0195	0.0015	LoaderFlags	0.0006	0.0003
BaseOfCode	0.0172	0.0013	MajorOperatingSystemVersion	0.0004	0.0001
SizeOfHeapReserve	0.0118	0.0015	MinorOperatingSystemVersion	0.0003	0.0002
MinorOperatingSystemVersion	0.0111	0.00047	MinorImageVersion	0.0002	0.0000
MajorOperatingSystemVersion	0.0051	0.0005	SizeOfStackCommit	0.0002	0
SizeOfStackCommit	0.0013	0.0001	BaseOfCode	0.0001	0.00015
MinorImageVersion	0.0012	0.0001	SizeOfStackReserve	0.0001	0.00015
SizeOfUninitializedData	0.0009	0.0002	SizeOfHeapReserve	0.0000	0.00020
e_sp	0.0002	0	SizeOfUninitializedData	0.0000	0.00008

The measurement in feature selection is used to know how parameters affect the performance, as shown in Table 3. From the measurement, classification will be performed using only the intersection at the top 5 influential parameters algorithm used for feature selection. Feature importance techniques allow us to focus on the data's most relevant and informative aspects, improve model interpretability, address noise and collinearity issues, and enhance computational efficiency, ultimately leading to better model performance.

4.3. Classifier

Results from the classifier indicate a comparison of classifiers using the combined ClaMP + Honeypot dataset and the ClaMP dataset with Decision Tree; both datasets show similar performance, with an accuracy of around 97.9% and an F1-score of 98.2%. The precision, recall, and F1-score are consistent across the two datasets.

Random Forest shows ClaMP + Honeypot dataset yields a slightly higher accuracy of 97.7% compared to 98.3% for the ClaMP dataset only. The ClaMP dataset exhibits slightly higher precision, recall, and F1-score values. The Support Vector Machine classifier is lower compared to the other two classifiers. Both datasets show similar results, with an accuracy of around 83.7% - 84.3% and an F1-score of 86.1% - 87.9%. The precision and recall scores are also lower than the other classifiers, as shown in Table 4.

In conclusion, the Decision Tree and Random Forest classifiers demonstrate consistent and high performance across both datasets. The Support Vector Machine classifier performs relatively lower accuracy, precision, recall, and F1-score. Therefore, the Decision Tree and Random Forest classifiers are recommended for malware detection using the given datasets.

4.4. Model

The Scikit-learn library from Python is used to conduct the analysis based on the experiment. Machine learning is used to detect malware on the Portable Executable (PE) dataset. Models' performance is assessed on preprocessed data, divided into testing and training subsets. Various metrics such as model accuracy, recall, precision, and F1-score are computed for each classifier.

Table 4. Comparison machine learning technique

Algorithm	Matrix	ClaMP + Honeypot	ClaMP
Decision Tree	Accuracy	97.9%	98.2%
	Precision	98.6%	98.3%
	Recall	97.9%	98.2%
	F1-score	98.2%	98.3%
Random Forest	Accuracy	97.7%	98.3%
	Precision	97.9%	98.2%
	Recall	98.4%	98.6%
	F1-score	98.1%	98.4%
Support Vector Machine	Accuracy	87.1%	90.3%
	Precision	90.1%	91.1%
	Recall	95.2%	96%
	F1-score	85.6%	86.7%

Table 5. Experiment result using parameters reduction

Algorithm	Matrix	ClaMP + Honeypot	ClaMP
Decision Tree	Accuracy	98.9%	99.2%
	Precision	99.2%	98.8%
	Recall	99.1%	99.6%
	F1-score	99.1%	99.2%
Random Forest	Accuracy	99.2%	99.4%
	Precision	99.1%	99.2%
	Recall	**99.6%**	**99.6%**
	F1-score	99.36%	99.4%
Support Vector Machine	Accuracy	87.1%	90.3%
	Precision	90.1%	91.1%
	Recall	95.2%	96%
	F1-score	85.6%	86.7%

Table 6. Gap result from parameters reduction

Algorithm	Matrix	ClaMP + Honeypot	ClaMP
Decision Tree	Accuracy	0.1%	0.1%
	Precision	0.6%	-0.5%
	Recall	1.2%	1.4%
	F1-score	1.1%	0.9%
Random Forest	Accuracy	1.5%	1.1%
	Precision	0.2%	0%
	Recall	0.8%	0.8%
	F1-score	1.26%	1%
Support Vector Machine	Accuracy	0%	0%
	Precision	0%	0%
	Recall	0%	0%
	F1-score	0%	0%

The experimental results for all classifiers are presented in Table 5. The decision tree classifier achieves an accuracy of 97.9%, recall of 98.6%, precision of 97.9%, and an F1-score of 98.2%. In comparison, the Random Forest classifier achieves an accuracy of 97.7%, recall of 98.4%, precision of 97.9%, and an F1-score of 98.1%. The Support Vector Machine (SVM) classifier achieves an accuracy of 83.7%, precision of 81.4%, recall of 95.5%, and an F1-score of 87.9%.

4.5. Evaluation

Results from the experiment and comparison with other machine learning techniques are provided in Table 4, which presents the model's performance. Given that the dataset has already been split into training and testing datasets, the model is constructed using the training data. The validation dataset is created by randomly selecting 10% of the training dataset to monitor the learning progress. Based on the evaluation results presented in Table 4, the Decision Tree and Random Forest algorithms performed better compared to the Support Vector Machine algorithm.

Both Decision Tree and Random Forest achieved high accuracy, precision, recall, and F1-score values, indicating their effectiveness in the classification task. On the other hand, the Support Vector Machine algorithm had lower accuracy, precision, and F1-score values, suggesting that it may not be as suitable for this specific classification problem.

The metrics presented in Table 5 provide a comprehensive analysis of the predictions made by the proposed model on the test dataset, considering five different parameters. These metrics offer insights into the performance and accuracy of the model in predicting each label accurately, and metrics, as shown in Table 6, perform gap results between models that use parameter reduction and models without parameter reduction. Decision Tree updated metrics show an improvement in accuracy, precision, recall, and F1-score compared to the previous data, indicating better performance. Random Forest updated metrics for Random Forest also demonstrate improvement across all performance measures. The utilization of these five parameters leads to a higher accuracy percentage compared to previous studies. These parameters are obtained through feature selection techniques, enabling the measurement of the dataset's feature importance.

5. Conclusion

The proposed model in this study presents a static analysis approach for malware detection, utilizing four layers: data acquisition, preprocessing, prediction, and performance evaluation.

The model demonstrates the ability to detect malware without executing the executable file, which enhances security and reduces potential risks. By employing a supervised machine learning approach with binary classification, the model effectively categorizes the dataset into either malicious or benign classes.

Various metrics such as F1-score, precision, recall, and accuracy are utilized to evaluate the proposed system's performance. The experimental results indicate that the Random Forest algorithm performs exceptionally well, outperforming other algorithms in terms of accuracy. The Random Forest algorithm achieves an impressive 99.20% accuracy rate in detecting malware, showcasing its effectiveness in identifying malicious samples. These findings suggest that the proposed model can serve as a reliable solution for malware detection, providing high accuracy and robust performance. It offers a non-execution-based approach, making it suitable for real-time detection and ensuring the security of systems and networks.

Acknowledgement

This research paper was conducted as part of the thesis research for the graduate program at Bina Nusantara University. The authors would like to express their sincere gratitude and appreciation to all those who have contributed to the completion of this paper. They hope this work will serve as a foundation for future research and projects, aiming to achieve further advancements and discoveries.

References

[1] Hassan Naderi et al., "Malware Signature Generation Using Locality Sensitive Hashing," *Communications in Computer and Information Science*, pp. 115–124, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[2] Attacks Every Few Seconds: Around 100 Malware Variants Per Minute Threaten IT Security, 2023. [Online]. Available: https://presse.gdata.de/news--attacks-every-few-seconds-around-100-malware-variants-per-minute-threaten-it-security?id=174381&menueid=28982&l=english

[3] Vasileios Kouliaridis et al., "A Survey on Mobile Malware Detection Techniques," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 2, pp. 204–211, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[4] Gerardo Fernandez, Lesson Learned from 2022, 2023. [Online]. Available: https://blog.virustotal.com/2023/01/lessons-learned-from-2022.html

[5] Leyi Shi et al., "Dynamic Distributed Honeypot Based on Blockchain," *IEEE Access*, vol. 7, pp. 72234–72246, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Seungjin Lee et al., "Classification of Botnet Attacks in IoT Smart Factory using Honeypot Combined with Machine Learning," *PeerJ Computer Science*, vol. 7, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Cheng Huang et al., "Automatic Identification of Honeypot Server Using Machine Learning Techniques," *Security and Communication Networks*, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[8] Rejwana Islam et al., "Android Malware Classification using Optimum Feature Selection and Ensemble Machine Learning," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 100–111, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[9] Hemashu Kamboj, and Gurpreet Singh, "Fake Access Point Detection and Prevention Techniques," *International Journal of P2P Network Trends and Technology*, vol. 3, no. 2, pp. 34-36, 2013. [Google Scholar] [Publisher Link]

[10] Md. Haris Uddin Sharif et al., "Comparative Study of Prognosis of Malware with PE Headers Based Machine Leaning Techniques," *2023 International Conference on Smart Computing and Application*, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Muhammad Shairoze Malik, "The Machine Learning in Malware Detection," *International Journal for Electronic Crime Investigation*, vol. 5, no. 3, 2021. [CrossRef] [Publisher Link]

[12] Iik Muhamad Malik Matin, and Budi Rahardjo, "Malware Detection Using Honeypot and Machine Learning," *2019 7th International Conference on Cyber and IT Service Management*, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[13] Sonali Tidke, and Pravin Karde, "Design Methodology of Botnet Attack for Smartphone," *SSRG International Journal of Computer Science and Engineering*, vol. 2, no. 5, pp. 11-15, 2015. [CrossRef] [Publisher Link]

[14] Ravi Kiran Varma Penmatsa, Akhila Kalidindi, and S. Kumar Reddy Mallidi, "Feature Reduction and Optimization of Malware Detection System Using Ant Colony Optimization and Rough Sets," *International Journal of Information Security and Privacy*, vol. 14, no. 3, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[15] Tobias Wuchner et al., "Leveraging Compression-Based Graph Mining for Behavior-Based Malware Detection," *IEEE Transactions Dependable Secure Computing*, vol. 16, no. 1, pp. 99–112, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[16] Abrar Hussain et al., "Malware Detection Using Machine Learning Algorithms for Windows Platform," *Lecture Notes in Networks and Systems*, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[17] Ajit Kumar, K.S. Kuppusamy, and G. Aghila, "A Learning Model to Detect Maliciousness of Portable Executable using Integrated Feature Set," *Journal of King Saud University - Computer and Information Sciences*, vol. 31, no. 2, pp. 252–265, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[18] Ryandy Djap et al., "XB-Pot: Revealing Honeypot-based Attacker's Behaviors," *2021 9th International Conference on Information and Communication Technology*, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[19] K. Iswarya, "Security Issues Associated With Big Data in Cloud Computing," *SSRG International Journal of Computer Science and Engineering*, vol. 1, no. 8, pp. 1-5, 2014. [CrossRef] [Publisher Link]

[20] Daniel Gibert, Carles Mateu, and Jordi Planes, "The Rise of Machine Learning for Detection and Classification of Malware: Research Developments, Trends and Challenges," *Journal of Network and Computer Applications*, vol. 153, p. 102526, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[21] Davide Chicco, and Giuseppe Jurman, "The Advantages of the Matthews Correlation Coefficient (MCC) Over F1 Score and Accuracy in Binary Classification Evaluation," *BMC Genomics*, vol. 21, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[22] Muhammad Ijaz, Muhammad Hanif Durad, and Maliha Ismail, "Static and Dynamic Malware Analysis Using Machine Learning," *2019 16th International Bhurban Conference on Applied Sciences and Technology*, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[23] Syed Khurram Jah Rizvi et al., "PROUD-MAL: Static Analysis-based Progressive Framework for Deep Unsupervised Malware Classification of Windows Portable Executable," *Complex & Intelligent Systems*, vol. 8, pp. 673–685, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[24] Mozammel Chowdhury, Azizur Rahman, and Rafiqul Islam, "Malware Analysis and Detection Using Data Mining and Machine Learning Classification," *Advances in Intelligent Systems and Computing*, pp. 266–274, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[25] Akram M. Radwan, "Machine Learning Techniques to Detect Maliciousness of Portable Executable Files," *2019 International Conference on Promising Electronic Technologies*, 2019. [CrossRef] [Google Scholar] [Publisher Link]